

Requirement verification in simulation-based automation testing

Eero Siivola¹, Seppo Sierla², Hannu Niemistö¹, Tommi Karhela^{1,2} and Valeriy Vyatkin^{2,3}

¹VTT Technical Research Centre of Finland Ltd

²Dept. of Electrical Engineering and Automation, Aalto University, Espoo, Finland

³SRT, Luleå Tekniska Universitet, Luleå, Sweden

E-mail: {eero.siivola, seppo.sierla}@aalto.fi, {hannu.niemisto, tommi.karhela}@vtt.fi, vyatkin@ieee.org

Abstract—The emergence of the Industrial Internet results in an increasing number of complex temporal interdependencies between automation systems and the processes to be controlled. There is a need for verification methods that scale better than formal verification methods and which are more rigorous than testing. Simulation-based runtime verification is proposed as such a method, and an application of metric temporal logic is presented as a contribution over the state-of-the-art work by the Modelica community, which is based on linear temporal logic that cannot capture several kinds of requirements that occur frequently in industrial process control systems. The practical scalability of the proposed approach is validated against a production process designed by an industrial partner, resulting in the discovery of several requirement violations. The scalability implications of alternative approaches for requirements formalization are also investigated theoretically from a computational complexity perspective for the requirements in our case study, and a broader investigation is proposed as further research.

I. INTRODUCTION

The emergence of the Industrial Internet results in an increasing number of cyber-physical dependencies between the automation system and the process to be controlled; in particular, the increasingly numerous and intelligent connected devices introduce complex temporal interdependencies that require advanced verification methods. There are multiple verification approaches, but traditionally three main techniques have been considered: theorem proving, model checking, and testing [1]. Theorem proving is based on formal mathematical proof of correctness [2]. Model checking is a computer-aided static method, in which the system model is checked against requirements using an exhaustive search over the entire problem space [3]. Theorem proving and model checking do not scale very well to big systems, whereas complete testing of a system would require creating test cases that cover all possible input and output combinations and this would require more test runs than would often be practical [4]. Runtime verification was developed to be a trade-off between model checking and testing to solve these problems [1], and it is possible to exploit a simulation model built in the system design phase also for runtime verification, as will be done in our case study.

In the Modelica community, runtime verification of requirements written in linear temporal logic (LTL) has recently been supported [5]. However, LTL cannot capture requirements

frequently found in industrial automation systems, e.g: after pressure exceeds a threshold, a safety valve must be opened within 2s. LTL does not support metricity of time, so it could only be used to verify that the safety valve will eventually be opened. Metric temporal logic (MTL) provides support for the metricity of time and thus has the capability to capture the said requirement. The research goal of this paper is to apply MTL to runtime verification of automation systems and to demonstrate it with an industrial case in mineral processing.

This paper is structured as follows. Section II reviews related research in simulation based verification, requirement formalization and runtime verification. Section III presents the theoretical approach. Section IV presents the case study, its implementation in a simulation environment and simulation results. Section V presents a discussion of the scalability of the proposed approach from the perspective of computational complexity. Section VI concludes the paper and identifies problems for further research.

II. LITERATURE REVIEW

A. Research goals

Simulation-based testing is one active area for verification of industrial automation systems. Research on simulation environments integrating the automation system and plant covers co-simulation of automation system and plant [6], tri-simulation including also the communications [7], hardware in-the-loop and software in-the-loop testing [8], scalability issues [9], interoperability [10] and multidomain plant models [11]. This body of research provides opportunities for various simulation-based verification approaches. Examples of simulation-based testing include interfacing programmable logic controller (PLC) application to simulation model of plant for virtual commissioning [12] and verification of smart grid automation systems under cyber-attacks targeted on sensors [13] and smart meters [14]. However, such methods lack a rigorous capture and validation of real-time requirements that is possible through temporal logic. LTL has been applied in the context of timed Petri nets [15] and PLC program verification [16], and an extension fuzzy temporal logic context free supports the formalizing of real-time requirements typically found in industrial process control [17], but these approaches do not support simulation-based runtime verification. In this paper, simulation-based runtime verification of real-time temporal

logic requirements is supported, and the overall research goal in section I can now be elaborated to the following research goals:

- to be able to perform runtime verification of industrial automation systems before the physical system exists
- to exploit the same simulation model in design, verification and operation phase activities, such as tracking simulation [18]
- to achieve scalability of the approach to an industrial-scale processes
- to investigate the limitations of simulation based runtime verification based on an industrial scale case study

B. Requirement formalization

Consider the following example requirements: “*If an elevator is a service lift, then it does not need to have an emergency phone*”. These kind of requirements can be formalized using the language of propositional logic [19]. Even though formalizing requirements is easy with propositional logic, this language is not expressive enough for industrial automation applications, in which behavior is time dependent. Different temporal logics have been constructed since 1970s to solve the problem of truth values changing in time (the concept was introduced by Pnueli *et al.* in [20]). Generally these temporal logics extend propositional logic with operators that allow expressing temporal relations of events [19]. Most commonly it is necessary to introduce only two new operators to be able to express temporal relations of events, resulting in the possibility to formalize requirements of the type:

- “*The elevator must never move if the door is open*”, or
- “*If the ‘close door’ -button is pushed, the door will eventually close*”

In order to verify such requirements, it is necessary to make temporal operators bounded in time. In some temporal logics a requirement is considered to be satisfied if it is true at every time instance of given timed state sequence [19].

Several other methods for formalizing requirements exist. Controlled natural languages (CNLs) are based on some natural language and preserve most of its natural properties but have more restrictive syntax, lexicon or semantics. CNLs can be divided to two major types: the ones improving readability and the ones that enable automatic processing, with few CNLs build on top of some temporal logic [21]. Complicated requirements written in pure temporal logic are long and difficult to understand, so the advantage of such CNLs is that they provide natural language templates can be used to interpret the most common structures of temporal logic. Property Specification Language (PSL) is an example of this kind of language [35]. Dwyer *et al.* ([22]) have collected over 500 examples of requirements from various engineering fields and their experience shows that 92% of them matched to one of their property specification templates and 80% of requirements matched to three most common patterns. Some limited domain requirements can also be formalized using domain specific languages (DSLs). Domain-specific embedded languages (DSELs) are a subset of DSLs that built on

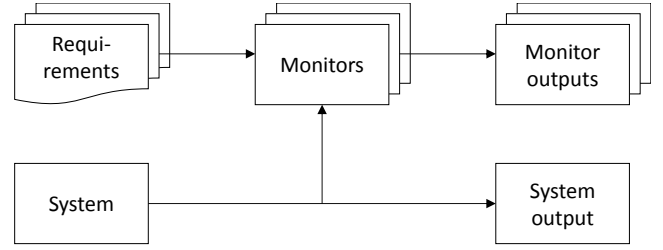


Fig. 1: Information flow in runtime verification.

some modular and extensible language like Haskell or ML but rely more on reuse of syntax and semantics than DSLs [23]. Using DSLs and especially DSELs has become more widespread in the field of model driven development (MDD) and the results in readability and efficiency of implemented DSLs are promising [24][25]. This approach for requirement formalization is quite appealing in some applications as it makes it possible to design the model and the requirements in very similar way. For example in Modelica, which is a dynamic simulation environment for industrial applications, research has been done about using DSLs for requirements formalization [26].

C. Runtime verification

Runtime verification is a lightweight and dynamic verification method used to determine whether a single finite run of a system violates given requirements [27]. A single finite run is a finite sequence of system states which, at a given instant in time, define the characteristics of the system [28]. The violation of requirements is checked with monitors, which are devices that read a set of states of a finite run and output a verdict [29]. A verdict is most typically a value from a truth domain, but the most familiar $\{true, false\}$ is not sufficient. For example, consider the requirement “*B must happen in less than 10 seconds after A*”. If A has happened, less than 10 seconds has passed, B has not happened and the run is over, it is not possible to output a true or false verdict. To emphasize the fact that requirements are only true for that current run, not universally, a truth domain can be formulated as $\{not\ violated, violated, not\ evaluated\}$ [3].

The information flow in runtime verification is shown in Fig 1. The monitors are automatically created from the temporal logic requirements. There is no one common way for creating monitors [30], but there are two methods for monitoring temporal requirements that have been studied by several researchers and a supported by more than one scientific or commercial tool [31]. These methods are based on formula rewriting and finite state machines. In rewriting based techniques, formalized requirements are updated at each time step during runtime. At each iteration every operator of the formalized requirement is rewritten to correspond to the current state of the system using predefined set of rules. After this the formulas are simplified to a canonical form using another set of rules [32][33]. Another approach is to create finite state machines from a temporal logic formula by going

through all possible evaluations using formula rewriting. By analyzing transitions between different formulas, a finite state machine can be created [30][34]. The state machine based method is only recommended when the monitored temporal logic formula is small and runtime overhead is desired to be minimal [33]; however, one reason to use runtime verification instead of other methods such as model checking is scalability, so formula rewriting is used in this paper.

III. THEORY

A. Metric Temporal Logic

The MTL formulation presented in this section mainly follows the formulation given by Thati *et al.* in [31] with one major change explained at the end of this section. Given a finite set P of propositions p , that evaluate to either true or false, MTL formulas are inductively defined as:

Definition III.1. Let ϕ be a MTL formula that can be any of the following:

$$\phi := p \mid \wedge \{\phi_1, \dots, \phi_n\} \mid \oplus \{\phi_1, \dots, \phi_n\} \mid \circ_I \phi \mid \phi_1 \mathcal{U}_I \phi_2$$

Here $n \in \mathbb{Z}_{\geq 0}$ and names of the operators above are: \wedge : and, \oplus : exclusive or, \circ_I : next, and \mathcal{U}_I : until. Also, in above $p \in P$, and I is one of the following:

- 1) An interval of non-negative real line. For a number x , $\pm I \pm x \equiv \{\pm y \pm x \mid y \in I\} \cap [0, \infty)$
- 2) A relative congruence $\approx_d c$ for real numbers $0 \leq c < d$. We define $y \approx_d c$ if and only if $y = c \bmod d$ and $\pm I \pm x \equiv \{y \mid y = \pm c \pm x \bmod d\}$
- 3) An absolute congruence $=_d c$ for real numbers $0 \leq c < d$. We define $y =_d c$ if and only if $y = c \bmod d$ and $\pm I \pm x \equiv \{y \mid y = c \bmod d\}$

A timed sequence ρ is a sequence of pairs $(x_i, \tau_i)_{0 \leq i \leq |\rho|}$ where x_i is the state at i^{th} time and τ_i is a non-decreasing sequence of times. For given timed state sequence ρ and index $1 \leq i \leq |\rho|$, let us define what it means for (ρ, i) to satisfy the formula ϕ . This can also be written as $(\rho, i) \models \phi$

$$\begin{aligned} (\rho, i) \models p & \quad \text{iff } p \in x_i \\ (\rho, i) \models \wedge \{\phi_1, \dots, \phi_n\} & \quad \text{iff } (\rho, i) \models \phi_i \text{ holds for all } 1 \leq i \leq n \\ (\rho, i) \models \oplus \{\phi_1, \dots, \phi_n\} & \quad \text{iff } (\rho, i) \models \phi_i \text{ holds odd number of times as } 1 \leq i \leq n \\ (\rho, i) \models \circ_I \phi & \quad \text{iff } i < |\rho|, (\rho, i+1) \models \phi \text{ and } t_{i+1} \in t_i + I \\ (\rho, i) \models \phi_1 \mathcal{U}_I \phi_2 & \quad \text{iff } (\rho, j) \models \phi_2 \text{ for some } j \text{ with } t_j \in t_i + I \text{ and } (\rho, k) \models \phi_1 \text{ for all } i \leq k < j \end{aligned}$$

Operators \wedge and \oplus as defined as a set operators for convenience as this way logical values *true* and *false* follow straight from the definitions of \wedge and \oplus . However, this notation

makes equations harder to read, so from this on let:

$$\begin{aligned} \text{true} &:= \wedge \{\} & \text{false} &:= \oplus \{\} \\ \phi &:= \wedge \{\phi\} & \phi &:= \oplus \{\phi\} \\ \phi_1 \oplus \dots \oplus \phi_n &:= \oplus \{\phi_1, \dots, \phi_n\} \\ \phi_1 \wedge \dots \wedge \phi_n &:= \wedge \{\phi_1, \dots, \phi_n\} \end{aligned}$$

Using the primitive operators defined above, it is possible to define other basic logic operators such as \vee : or, \rightarrow : implication, and \neg : not. Also, because temporal operators \circ_I and \mathcal{U}_I are not very intuitive, it often is useful to define additional temporal operators, the most useful of which are \Diamond_I : finally and \Box_I : globally. For these (ρ, i) satisfies formula ϕ :

$$\begin{aligned} (\rho, i) \models \phi_1 \vee \dots \vee \phi_n & \quad \text{iff } (\rho, i) \models \phi_i \text{ holds at least once as } 1 \leq i \leq n \\ (\rho, i) \models \phi_1 \rightarrow \phi_2 & \quad \text{iff } (\rho, i) \models \phi_1 \text{ and } (\rho, i) \not\models \phi_2 \text{ do not both hold} \\ (\rho, i) \models \Diamond_I \phi & \quad \text{iff } (\rho, i) \models \phi \text{ for some } j \text{ with } t_j \in t_i + I \text{ and } i \leq j \leq |\rho| \\ (\rho, i) \models \Box_I \phi & \quad \text{iff } (\rho, i) \models \phi \text{ for all } j \text{ with } t_j \in t_i + I \text{ and } i \leq j \leq |\rho| \end{aligned}$$

Which can be derived iteratively as follows:

Definition III.2. Let ϕ be an additional MTL formula, these can be derived using already defined MTL formulas as:

$$\begin{aligned} \phi_1 \vee \dots \vee \phi_n &= \neg((\neg\phi_1) \wedge \dots \wedge (\neg\phi_n)), \\ \phi_1 \rightarrow \phi_2 &= \neg(\phi_1 \wedge (\neg\phi_2)), \\ \neg\phi &= \text{true} \oplus \phi, \quad \Diamond_I \phi = \text{true} \mathcal{U}_I \phi, \quad \Box_I \phi = \neg(\Diamond_I(\neg\phi)) \end{aligned}$$

From this on let $\circ := \circ_{[0, \infty)}$, $\circ_{\leq t} := \circ_{[0, t]}$, $\circ_{> t} := \circ_{(t, \infty)}$ and similarly for other operators including I . As a side note the most basic temporal logic, LTL can now be defined to be MTL, for which all $I = [0, \infty)$.

In addition to these operators defined above, as discussed in the literature review, it often is useful to introduce some additional operator templates for the most popular requirements. Kansas University's Laboratory for Specification, Analysis, and Transformation of Software (SAnToS) maintains a web repository [35] for commonly used and well tested LTL patterns that can easily be extended to MTL. The most useful pattern in our research was noticed to be:

$$\begin{aligned} \text{timedTrigger}_I(\phi_1, \phi_2) &= \\ \Box \left(((\neg\phi_1) \wedge (\circ\phi_1)) \rightarrow \circ(\Diamond_I \phi_2) \right) \end{aligned}$$

The above template can be used to formalize two useful requirement. If $I = [0, \infty)$, the template corresponds to requirement "When ϕ_1 becomes true, ϕ_2 must eventually become true". If $I = (t_1, t_2]$, the template corresponds to requirement "When ϕ_1 becomes true, ϕ_2 must become true within t_1 to t_2 seconds"

Unlike in the formulation given by Thati *et al.* in [31] and original formulation of MTL by Alur *et al.* in [36], past time temporal operators that correspond to operators \circ and \mathcal{U} are not considered here to be part of MTL. Although it is known that introducing the past operators makes MTL formulas more expressive, no evident advantage from introducing them would have been gained at least for the requirements formalized in the test process [37]. Similar results from LTL, where only future time operators have been sufficient for presenting requirements can be found from [38] and [22]. Furthermore, leaving past time operators makes the implemented monitoring algorithm more efficient, as it is shown in the next section.

B. Term rewriting based runtime monitoring for MTL

Let $\phi\{\rho, i\}$ symbol the derivation of MTL formula ϕ with respect to the i^{th} event of timed state sequence ρ . This derivation process for different MTL formulas is iteratively defined as follows:

Definition III.3. Let ρ be timed state sequence and $1 \leq i \leq |\rho|$, $i \in \mathbb{N}$. Then:

$$\begin{aligned} p\{\rho, i\} &= p \in \pi_i \\ (\phi_1 \wedge \dots \wedge \phi_n)\{\rho, i\} &= \phi_1\{\rho, i\} \wedge \dots \wedge \phi_n\{\rho, i\} \\ (\phi_1 \oplus \dots \oplus \phi_n)\{\rho, i\} &= \phi_1\{\rho, i\} \oplus \dots \oplus \phi_n\{\rho, i\} \\ (\circ_I \phi)\{\rho, i\} &= \\ &\phi_1 \begin{cases} \phi & \text{if } \tau_{i+1} \in \tau_i + I \text{ and } i < |\rho| \\ false & \text{else.} \end{cases} \\ (\phi_1 \mathcal{U}_I \phi_2)\{\rho, i\} &= \left(\begin{cases} \phi_2\{\rho, i\} & \text{if } 0 \in I \\ false & \text{else} \end{cases} \right) \\ &\vee \left(\begin{cases} (\phi_1\{\rho, i\}) \wedge (\phi_1 \mathcal{U}_{I'} \phi_2) & \text{if } i < |\rho| \\ false & \text{else} \end{cases} \right) \\ &\text{here } I' = I - \tau_{i+1} + \tau_i \end{aligned}$$

In order to make the algorithm more efficient, canonical term-rewriting system for Boolean algebra developed by Hsiang in [39] can be used. This algorithm keeps the formula in the algebraic normal form where equivalent formulas have the same representation. The algorithm is defined as a term-rewriting system defined above. Because subformulas of the connectives are represented as sets, associativity and combination of subformulas with the same representation is automatically taken care of.

Definition III.4. Let ϕ be any MTL formula. Then the equations on the left side of \Rightarrow are transformed to the equations on its right side:

$$\begin{aligned} \phi \wedge false &\Rightarrow false & \phi \wedge \phi &\Rightarrow \phi & \phi \wedge true &\Rightarrow \phi \\ \phi \oplus false &\Rightarrow \phi & \phi \oplus \phi &\Rightarrow false \\ \phi_1 \wedge (\phi_2 \oplus \phi_3) &\Rightarrow (\phi_1 \wedge \phi_2) \oplus (\phi_1 \wedge \phi_3) \end{aligned}$$

The algorithm presented in this section mainly follows the one presented by Thati *et al.* in [31] with two exceptions. Firstly, the past operators are not included in this algorithm because, as explained in the previous section, no evident

advantage from using them would be gained. Secondly, the algorithm by Thati *et al.* transforms the MTL formulas to binary presentations and performs the evaluation and canonisation in that form. However, if only future time operators are used, then the binary presentation is not advisable, as it makes the algorithm more inefficient. Proof for this is presented in the next section.

C. Complexity of the algorithm

As mentioned in section III-A, monitoring algorithm is executed runtime and needs to be efficient. In this section, the time complexity of the monitoring algorithm is derived. The derivation mainly follows the proof presented in the article by Barringer *et al.* in [40] with one major change that is explained at the end of the section.

Let $\mathcal{F}(\phi)$ be the set of all subformulas of ϕ that are either atomic propositions (true, false and $p \in P$) or rooted at temporal operators (\mathcal{U}_I and \circ_I). Let us also define all subformulas derivable from $\mathcal{F}(\phi)$ at τ_j :

$$\begin{aligned} \mathcal{F}_j^+(\phi) &= \mathcal{F} \cup \{\phi_1 \mathcal{U}_{I'} \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2 \in \mathcal{F}(\phi), \\ &\quad I' = I + \tau_i - \tau_j \text{ for some } 0 \leq i \leq j\} \end{aligned}$$

Now using this notation, we can define the size of MTL formula at τ_j to be $m_j = |\mathcal{F}_j^+(\phi)|$. It must be noted that $\circ_I(\phi_i)$ evaluates only to either ϕ_i or false, so that is why $\circ_I(\phi_i)$ is not included in $\mathcal{F}_j^+(\phi)$. The sequence m_j cannot be bounded above in general. As a simple example, monitoring formula $\Diamond \Diamond_{\approx 2} \text{true}$ requires storage of all times τ_j that have occurred in the timed sequence so far. If times τ_i are integers as well as the divisors of the modulo operators, m_j has an upper bound that is independent of the timed sequence. If the modulo operators are not used, it is enough to require that the interval between subsequent times $(\tau_{i+1} - \tau_i)$ is bounded below.

Equations of Definition III.4 keep any MTL formula in a canonical form. Canonical forms are exclusive disjunctions of conjunctions, where the conjuncts are either propositions or temporal operators. Furthermore, after having propagated any MTL formula ϕ in the time using Definition III.3 $((\dots((\phi\{\rho, i\})\{\rho, i+1\})\dots)\{\rho, i+k\})$, the conjuncts are either propositions or temporal operators of $\mathcal{F}_j^+(\phi)$. Since there are at most m_j such subformulas at time τ_j , there are at most 2^{m_j} possibilities to combine them in a conjunction.

The space requirement of a conjunction is $\mathcal{O}(m \log(m))$, assuming that instead of storing the conjuncts, pointers to the conjuncts are stored that each take $\mathcal{O}(\log(m))$ space. The multiplier here must be m , since the number of operands of \wedge and \oplus operators is only bounded by m . Thus, the space requirement for storing any MTL formula structure derived from the original MTL formula ϕ is $\mathcal{O}(m 2^m \log(m))$. In addition to the structure, also the conjuncts appearing in the conjunction are needed to be stored. As size of each conjunction is $m \log(m)$ the space needed to store all conjuncts is $\mathcal{O}(m^2 \log(m))$. Hence the total space requirement is $\mathcal{O}(m 2^m \log(m) + m^2 \log(m)) = \mathcal{O}(m 2^m \log(m))$.

The time complexity of the MTL monitoring algorithm is bounded by the size of the MTL formula at each index of timed state sequence ρ . Since at all times the algorithm traverses every subformula of a MTL formula, it takes $\mathcal{O}(m2^m \log(m))$ just to go through the MTL formula. Also, at each subformula, the algorithm can spend $\mathcal{O}(m2^m \log(m))$ to do the conjunction and exclusive disjunction. Thus the time complexity of the algorithm is $\mathcal{O}(m2^m \log(m)) * \mathcal{O}(m2^m \log(m)) = \mathcal{O}(m^2 2^{2m} \log^2(m))$, which is more efficient than the $\mathcal{O}(m^2 2^{3m})$ algorithm by Thati *et al.* [31].

IV. CASE STUDY

A. Pressure leaching process

The implemented tool is tested on a simulation model of a real industrial mineral processing plant. The whole process, where copper, cobalt and zinc are extracted from thermal treated limestone, known as calcine, is unnecessary large to be used as a test process. Therefore only subprocess called pressure leaching is used. The whole process is described briefly in the next paragraph in order for the reader to find the context of chosen system. After this, pressure leaching is described in more detailed manner.

Calcine is first crushed, grinded and then mixed with water in the phase called pulping. The resulting solution called slurry is then led to the pressure leaching process where metals are dissolved into the solution and pure lime is extracted from it. The leach residue, pregnant leach solution (PLS), that has high concentration of metals is then processed in three subsequent phases in each of which one metal is extracted from the slurry. After this, all compounds are end processed in suitable ways resulting copper cathodes, cobalt carbonate, zinc carbonate and neutralised waste. Pressure leaching consists of four sequent subprocesses:

- 1) The calcine slurry is first heated and pressurized in a subprocess called autoclave feed and preheating. Oversized particles are rejected from the solution and it is fed to a feed tank. After this, slurry is heated first to 100 °C in atmospheric preheater (ATM preheater) and then to 130–170 °C and pressurized to absolute pressure of 3–7 bar in high pressure preheater (HP preheater). Simplified flow

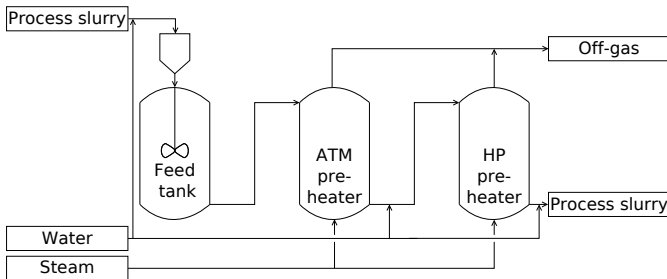


Fig. 2: The first subprocess of pressure leaching: calcine slurry is heated in two stages.

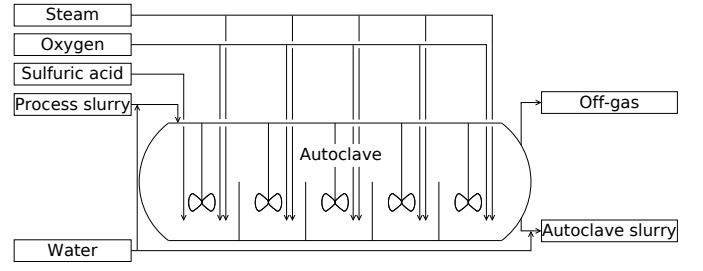


Fig. 3: The second subprocess of pressure leaching: metals dissolve to process solution inside an autoclave.

diagram of the feeding and preheating is illustrated in Fig 2.

- 2) In autoclave the chemical reactions happen and metals dissolve to the solution. Process slurry is fed to an autoclave that consists of five compartments each of which is equipped with oxygen input, steam input and agitator. The oxygen partial pressure is kept at 5 bar to maintain the oxidative conditions for reactions. The absolute pressure in autoclave is 28 bar. Steam is injected to maintain the temperature of 220 °C. In addition to these injections, sulphuric acid concentration of 40–50 g/L is maintained inside the autoclave by injecting it to the first compartment to keep the solution acidic enough for metals to dissolve. Simplified flow diagram of the autoclave is illustrated in Fig 3.
- 3) Slurry from the autoclave is fed by pressure difference to flashing. Flashing is done in two stages and its purpose is to decrease temperature and pressure of the slurry in a controlled manner. First in high pressure flash vessel (HP flash vessel), the pressure drops to 3–7 bar and temperature to 130–170 °C. After this in atmospheric flash vessel (ATM flash vessel), the pressure drops to absolute pressure of 1 bar and temperature to 100 °C. The off-gases of the flash vessels are used to warm the process slurry in the preheating subprocess. Also, to increase the efficiency of the whole process, heated slurry from the fourth subprocess is fed to the ATM flash vessel together with the slurry from HP flash vessel. Simplified flow diagram of the flashing is illustrated in Fig 4.

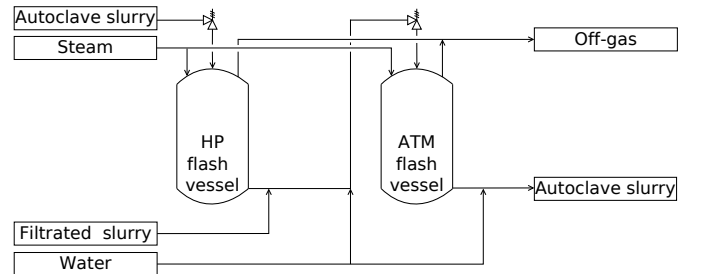


Fig. 4: The third subprocess of pressure leaching: temperature of calcine slurry is cooled in two stages.

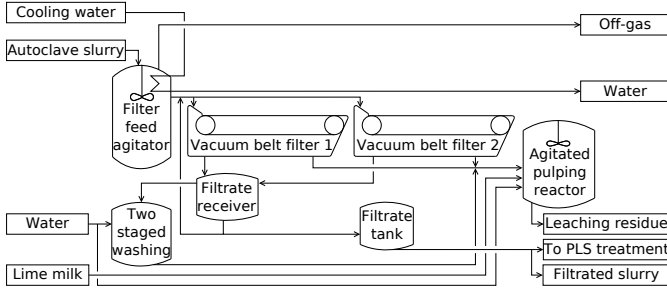


Fig. 5: The fourth subprocess of pressure leaching: calcine slurry is filtered.

4) From flashing the slurry is fed to leach residue filtration, which purpose is to separate most of the lime from the slurry. First the slurry is cooled in an filter feed agitator to 80 °C and then pumped to two vacuum belt filters. Acidic filtrate from vacuum belt filters, that is rich of metals, is collected to filtrate receiver. Part of the solution in filtrate receiver is channelled to filtrate tank, from where it is pumped further to PLS treatment and back to flashing. The rest is washed with fresh water in two staged washing. Washed filtrate is pumped to agitated pulping reactor. Also the residue that is left at the end of the vacuum belt filters is pumped to agitated pulping reactor. In agitated pulping reactor the residue is pulped with fresh water and simultaneously its pH is increased with lime milk. Simplified flow diagram of the filtration is illustrated in Fig 5.

The process and its automation were modelled using a software called Apros, which is built using software implementation platform called Simantics [41]. Simulation model of the first subprocess, autoclave feed and preheating is illustrated in Fig 6.

B. Elicited requirements

45 formalized requirements imposed on the automation of the process of the case study can be divided to the following three groups:

1) Limit requirements: This group contains some absolute limits that the process variables must never exceed. E.g. "Pressure of HP flash vessel must always be between absolute pressure of 3 and 7 bar". 34 of this kind of requirements were formalized. These requirements are related to the upper and lower limits of slurry in all tanks, temperatures and pressures in autoclaves, preheaters and flashers, and to mass flows in some pipes. These requirements can be formalized as:

$$\Box p$$

Here p is a function that returns true if the wanted state is within bounds.

2) Order related requirements: This group contains requirements for order of two events. These requirements can further be divided into two categories:

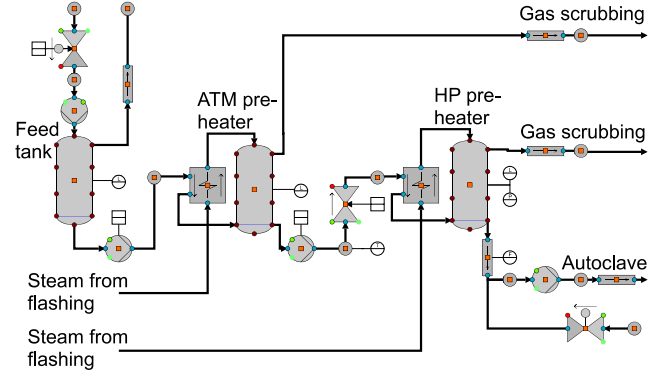


Fig. 6: Apros model of autoclave feed and preheating subprocess.

a) Latter event must happen within some time limit. E.g. "If the pressure difference between steam feed line and autoclave is less than 0.1 bar, the steam feed line valve must close within 60 seconds". 6 of this kind of requirements were formalized. Mostly these requirements are related to closing valves if the pressure difference between some tank and its feed line falls under some pre defined limit or slurry levels in tanks go under some predefined limit. These requirements can be formalized as:

$$\text{timedTrigger}_{[0, t]}(p_1, p_2)$$

Here t is the time limit, p_1 is a function returning true when the first event happens and p_2 is a function that returns true when the latter event happens.

b) The latter event must happen after a delay. E.g. "If the slurry level in Autoclave goes under the limit of 1 meter, then the output valve must not close before 30 seconds, but must be closed after 60 seconds". One of this kind of requirement was formalized. These requirements can be formalized as:

$$(\neg \text{timedTrigger}_{[0, t_1]}(p_1, p_2)) \wedge \text{timedTrigger}_{[t_1, t_2]}(p_1, p_2)$$

Here t_1 is the time limit before which the latter event should not happen, t_2 is the time limit before which it should happen, p_1 is a function returning true when the first event happens and p_2 is a function returning true when the latter event happens.

3) Various complex requirements: Requirements that don't fall into any of the other categories. 4 of this kind of requirements were formalized. One of these requirements states that if one of the following properties is true:

- Slurry level in the HP preheater is too high, p_1 .
- Pressure in the HP preheater is too high, p_2 .
- Temperature of the slurry entering HP preheater is too high, p_3 , continuously for more than 60 seconds.
- Slurry mass flow to the HP preheater is too high, p_4 , continuously for more than 60 seconds

Then the input valve of the HP preheater must be closed, p_5 , within 30 seconds. This requirement can be formalized as:

$$\text{timedTrigger}_{[0, 30]}(p_1 \wedge p_2 \wedge \Box_{[0, 60]}p_3 \wedge \Box_{[0, 60]}p_4, p_5)$$

C. Results

The selected tests and requirement verification results are presented in TABLE I. The results show that all tests violated at least 7 requirements. These requirements are same in all cases and are all related to the last subprocess, leach residue filtration as none of the requirements imposed on this part of the process could be satisfied.

In addition to these requirements that were violated in all cases, more knowledge of the automation can be gained from the test case runs that violate also other requirements. Small oscillations occur when the process is started and thus the slurry level requirements in the preheating subprocess are violated. Motor break-down after HP preheater causes the slurry level in the first two autoclave compartments to drop.

Autoclave control seems to be sensitive to malfunction of off-gas valve. Instantly when the malfunction occurs, the temperature inside all autoclave compartments starts to drop as the steam input valves are closed. Simultaneously partial pressure of the oxygen becomes too large and in addition to the 7 requirements that are never satisfied, 8 more requirements are violated.

Temperature measurement malfunction before ATM flash vessel causes the fresh water feed valve to close so that temperature would increase and thus the vessel starts to empty too fast before the later parts of the process start to react. Because of this, the temperature and liquid level requirements are violated.

V. DISCUSSION

The process used in the case study is a real industrial process. Thus the requirements imposed on the automation do not follow the state-of-the-art guidelines and best practices. One difference is related to the ambiguity of the requirements. Because of this, part of the requirements can be understood in several different ways. E.g. requirements of form "When ϕ_1 is true, then after a delay t_1 , within t_2 , ϕ_2 must be true". This requirement can be understood as a requirement of form:

$$(\neg \text{timedTrigger}_{[0, t_1]}(\phi_1, \phi_2)) \wedge \text{timedTrigger}_{[t_1, t_2]}(\phi_1, \phi_2)$$

or alternatively as:

$$\text{timedTrigger}_{[0, t_1]}(\phi_1, \Box_{[0, t_2]} \phi_2)$$

When understood latterly, problems occur with almost any t_2 . $\Box_{[0, t_2]} \phi_2$ can become true any moment $[0, t_1]$ after ϕ_1 has become true. Thus, after ϕ_2 has become true, the evaluated formula starts to store formulas $\Box_I \phi_2$ that are rooted to $\Box_{[0, t_2]} \phi_2$. Because this happens, after k simulation steps, the evaluated MTL formula contains a subformula of form

$$(\Box_I \phi) \vee (\Box_{I-\tau_1} \phi) \vee \dots \vee (\Box_{I-\tau_1 \dots - \tau_k} \phi)$$

TABLE I: Description of performed tests, what they are testing and number of requirements violations.

Test description	Tested subprocess control	Violations
Simulation ran normally	All controls	7
Simulation model started	All controls	9
Large deviation added to the level measurement of the buffer tank	Buffer tank control	7
Temperature measurement failure before HP preheater	Preheating control	7
Motor breakdown after HP preheater	Preheating control	9
Deviation in the pressure measurement of the first compartment	Autoclave control	7
Malfunction in the oxygen feed valve of the first compartment	Autoclave control	7
Autoclave temperature measurement failure in the second compartment	Autoclave control	8
Large pressure drop in the steam feed valve of the third compartment	Autoclave control	7
Malfunction in the autoclave off-gas valve	Autoclave control	15
Bias added to the level measurement in the last compartment	Autoclave control	9
Temperature measurement failure before ATM flash vessel	Flashing control	9
Motor breakdown after ATM flash vessel	Flashing control	7

Where $I = [0, t_2]$ and τ_i 's, are simulation step lengths for simulation steps $1 \leq i \leq k$. It is possible to solve the described problem by creating a set of simplification rules for temporal operators $\phi_1 \mathcal{U}_I \phi_2$. By using one De Morgan's law and the definition of \mathcal{U}_I , it can be proved that:

$$(\Box_I \phi) \vee (\Box_{I-\tau_1} \phi) \vee \dots \vee (\Box_{I-\tau_1 \dots - \tau_k} \phi) = \Box_{I-\tau_1 \dots - \tau_k} \phi$$

VI. CONCLUSIONS AND FURTHER WORK

When the requirements were understood in the way that \Box_I 's cause problems, the monitoring algorithm became considerably slower. In addition to this, ready solutions to extend the implemented algorithm to simplify this kind of MTL formulas could not be found from the literature. Because of this, the requirements were implemented so that the run-time complexity of the monitors did not cause problems. The above example is only one kind of requirement that is computationally intensive and thus limits the scalability to real industrial processes. A systematic search for other kinds of computationally heavy requirements and the development of simplification rules for them are a topic for further work.

The requirements verification method could reveal errors in the automation of the process even though the test case selection was not based on process knowledge but rather on typical use cases and some of the most probable error sources. Also, it could reveal requirement violations caused

by interconnections of subprocesses, which are hard to find with traditional automation testing methods. The implemented requirement monitors did not slow down the simulation even when all 45 monitors were ran simultaneously. No requirements impossible to formalize with MTL were found in this case study. In addition to this, most of the requirements could be grouped similarly as found by Dwyer *et al.* in [22].

ACKNOWLEDGMENT

This research has been done as a part of S-STEP project, which is funded by Finnish Metals and Engineering Competence Cluster (FIMECC). The authors would like to thank Juha Kortelainen and Tuomas Miettinen from VTT Technical Research Centre of Finland Ltd.

REFERENCES

- [1] M. Leucker, and C. Schallhart, "A Brief Account of Runtime Verification", *The Journal of Logic and Algebraic Programming*, vol. 78, pp. 293–303, Feb. 2009.
- [2] O. Balci, "Principles and Techniques of Simulation Validation, Verification, and Testing", *Simulation Winter Conf.*, Arlington, TX, 1995, pp. 147–154.
- [3] W. Schamai, "Model-Based Verification of Dynamic System Behavior Against Requirements: Method, Language, and Tool", Ph.D. dissertation, Dept. Comput. and Inform. Sci., Linköping University, Linköping, Sweden, 2013.
- [4] G. Myers, C. Sandler, and T. Badgett, "The Psychology and Economics of Program Testing" in *The Art of Software Testing*, 2nd ed., New Jersey: John Wiley & Sons, 2004, pp. 5–20.
- [5] M. Otter *et al.*, "Formal Requirements Modeling for Simulation-Based Verification", *Int. Modelica Conf.*, Versailles, France, 2015, pp. 625–635.
- [6] C.-H. Yang, G. Zhabelova, C.-W. Yang, and V. Vyatkin, "Cosimulation Environment for Event-Driven Distributed Controls of Smart Grid", *IEEE Trans. Ind. Informat.*, vol. 9, pp. 1423–1435, Aug. 2013.
- [7] N. Kashyap, C.-W. Yang, S. Sierla, P.G. Flikkema, "Automated Fault Location and Isolation in Distribution Grids with Distributed Control and Unreliable Communication", *IEEE Trans. Ind. Electron.*, vol. 62, pp. 1–8, Dec. 2014.
- [8] J. Wang, *et al.*, "Development of a Universal Platform for Hardware In-the-Loop Testing of Microgrids", *IEEE Trans. Ind. Informat.*, vol. 10, pp. 2154–2165, Nov. 2014.
- [9] K. Anderson, J. Du, A. Narayan and A. El Gamal, "GridSpice: A Distributed Simulation Platform for the Smart Grid", *IEEE Trans. Ind. Informat.*, vol. 10, pp. 2354–2363, Nov. 2014.
- [10] G. Manassero, E.L. Pellini, E.C. Senger, and R.M. Nakagomi, "IEC61850Based Systems Functional Testing and Interoperability Issues", *IEEE Trans. Ind. Informat.*, vol. 9, pp. 1436–1444, Aug. 2013.
- [11] C. Molitor, S. Gross, J. Zeitz, and A. Monti, "MESCOA Multienergy System Cosimulator for City District Energy Systems", *IEEE Trans. Ind. Informat.*, vol. 10, pp. 2247–2256, Nov. 2014.
- [12] H. Carlsson, B. Svensson, F. Danielsson, and B. Lennartson, "Methods for Reliable Simulation-Based PLC Code Verification", *IEEE Trans. Ind. Informat.*, vol. 8, pp. 267–278, May 2012.
- [13] H. Jinping *et al.*, "Sparse Malicious False Data Injection Attacks and Defense Mechanisms in Smart Grids", *IEEE Trans. Ind. Informat.*, vol. 11, pp. 1198–1209, Oct. 2015.
- [14] R. Deng, G. Xiao, and R. Lu, "Defending Against False Data Injection Attacks on Power System State Estimation", *IEEE Trans. Ind. Informat.*, to be published, DOI: 10.1109/TII.2015.2470218.
- [15] M. Kloetzer, C. Mahulea, C. Belta, and M. Silva, "An Automated Framework for Formal Verification of Timed Continuous Petri Nets", *IEEE Trans. Ind. Informat.*, vol. 6, pp. 460–471, Aug. 2010.
- [16] B.F. Adiego *et al.*, "Applying Model Checking to Industrial-Sized PLC Programs", *IEEE Trans. Ind. Informat.*, vol. 11, pp. 1400–1410, Dec. 2015.
- [17] J. Perez *et al.*, "FTL-CFree: A Fuzzy Real-Time Language for Runtime Verification", *IEEE Trans. Ind. Informat.*, vol. 10, pp. 1670–1683, Aug. 2014.
- [18] G.S. Martinez *et al.*, "A hybrid approach for the initialization of tracking simulation systems", *Conference on Emerging Technologies & Factory Automation*, Luxembourg, 2015, pp. 1–8.
- [19] B. Bellini, R. Mattolini, and P. Nesi, "Temporal Logics for Real-Time System Specification", *ACM Computing Surveys*, vol. 32, pp. 12–42, Mar. 2000.
- [20] A. Pnuelli, "The Temporal Logic of Programs", *Annual Symposium on Foundations of Computer Science*, Providence, RI, 1977, pp. 46–57.
- [21] T. Kuhn, "A Survey and Classification of Controlled Natural Languages", *Computational Linguistics*, vol. 40, pp. 121–170, Mar. 2014.
- [22] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, "Patterns in Property Specifications for Finite-State Verification", *International Conference on Foundations of Software Engineering*, Los Angeles, USA, 1999, pp. 411–420.
- [23] P. Hudak, "Modular Domain Specific Languages and Tools", *International Conference on Software Reuse*, Victoria, Canada, 1998, pp. 134–142.
- [24] M. Mernik, and J. Heering, "When and How to Develop Domain-Specific Languages", *ACM computing surveys*, vol. 35, pp. 316–344, Dec. 2005.
- [25] J.S. Cuadrado, and J.G. Molina, "Building Domain-Specific Languages for Model-Driven Development", *IEEE Softw.*, vol. 24, pp. 48–55, Sept. 2007.
- [26] A. Tundis, L. Rogovchenko-Buffoni, P. Fritzson, and A. Garro, "Modeling System Requirements in Modelica: Definition and Comparison of Candidate Approaches", *International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Nottingham, UK, 2013, pp. 15–24.
- [27] S. Colin, and L. Mariani, "Run-Time verification" in *Model-Based Testing of Reactive Systems* Heidelberg: Springer, 2005, pp. 525–555.
- [28] *IEEE standard glossary of software engineering terminology*, standard 610.12, 1990.
- [29] J. Levy, H. Saïdi, and T.E. Uribe, "Combining Monitors for Runtime System Verification", *Electronic Notes in Theoretical Computer Science*, vol. 70, pp. 112–127, Dec. 2002.
- [30] J. Delgado, A. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools", *IEEE Trans. Softw. Eng.*, vol. 30, pp. 859–872, Dec. 2004.
- [31] P. Thati, and G. Roşu, "Monitoring Algorithms for Metric Temporal Logic Specifications", *Electronic Notes in Theoretical Computer Science*, vol. 113, pp. 145–162, Jan. 2005.
- [32] K. Havelund, and G. Roşu, "Monitoring Programs Using Rewriting", *International Conference on Automated Software Engineering*, San Diego, CA, 2011, pp. 135–143.
- [33] K. Havelund, and G. Roşu, "An Overview of the Runtime Verification Tool Java PathExplorer", *Formal methods in system design*, vol. 24, pp. 189–215, Mar. 2004.
- [34] Y. Falcone, and L.D. Zuck, "Runtime Verification: the Application Perspective", *International Journal on Software Tools for Technology Transfer*, vol. 17, pp. 121–123, Apr. 2015.
- [35] The Laboratory for Specification, Analysis, and Transformation of Software, "Spec Patterns", Available: <http://patterns.projects.cis.ksu.edu/>.
- [36] R. Alur, and T.A. Henzinger, "Real-Time Logics: Complexity and Expressiveness", *Symposium on Logic in Computer Science*, 4-7.6.1990, Philadelphia, PA, 1990, pp. 390–401.
- [37] P. Boyer, F. Chevalier, and N. Markey, "On the Expressiveness of TPTL and MTL", *International conference on Foundations of Software Technology and Theoretical Computer Science*, Hyderabad, India, 2005, pp. 97–116.
- [38] T. Tommila, and A. Pakonen, "Controlled Natural Language Requirements in the Design and Analysis of Safety Critical I&C Systems", Technical Research Centre of Finland Ltd, Research Report VTT-R-01067-14, Mar. 2014.
- [39] J. Hsiang, "Refutational Theorem Proving Using Term-Rewriting Systems", *Artificial Intelligence*, vol. 25, pp. 255–300, Mar. 1985.
- [40] H. Barringer, and A. Goldberg, "EAGLE Does Space Efficient LTL Monitorings", University of Manchester, Department of Computer Science, Pre-Print CSPP-25, 2003.
- [41] T. Karhela, A. Villberg, and H. Niemistö, "Open Ontology -Based Integration Platform for Modeling and Simulation in Engineering", *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 3, pp. 1–36, Jun. 2012.